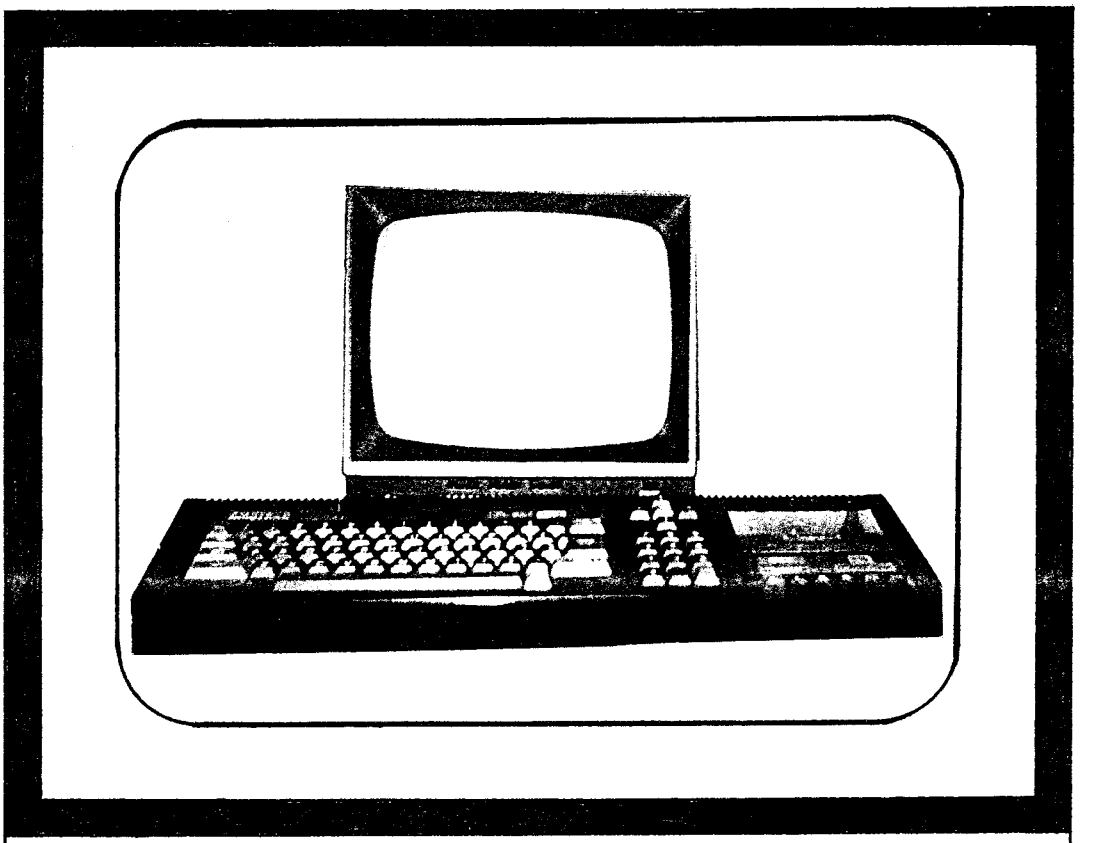
PRINT—OUT

Issue Three

Price 70p



BY Thomas Defoe, Mark Gearing and Jonathan Haddock

Contributor :- Bob Taylor

Including :-

BASIC & M/C tutorials

LINECHECKER—program

Homebrew Software

Technical Tips

INDEX

Miscellaneous

Page 3 - EDITORIAL - A warm welcome !
Page 40 - OFFERS - Things to buy for your CPC
Page 41 - SMALL ADS - Readers' goods on sale

Features

Page 4 - TECHNICAL TIPS - Solutions to readers' queries
Page 20 - LETTERS - Your thoughts on the CPC

Reviews

Page 15 - HOMEBREW SOFTWARE - Adventures and simulations
Page 30 - GAMES SOFTWARE - The good and the bad

Programming

Page 6 - BEGINNER'S BASIC - Part 3 of this tutorial
Page 8 - IMPROVEMENTS - Modifications to a past program
Page 9 - LINECHECKER - Eliminate those typing errors
Page 18 - BITS AND PIECES - More useful snippets
Page 23 - MAZE - A perplexing game to type and play
Page 28 - BASIC DEPROTECTION - Break into BASIC
Page 33 - MACHINE CODE - Calculating in M/C
Page 37 - ADVANCED BASIC - Grasping BASIC commands

We would like to express our thanks to Mr Gearing and Black Horse Agencies, Januarys, for the continued use of their photocopier in the production of this issue.

Please note that we do not support piracy in any form whatsoever unless backups are for the sole use of the original owner.

Editorial→

WELCOME TO ISSUE THREE OF PRINT-OUT.

First of all we would like to apologize for the delay in producing this issue of the magazine. We hope that this has not been too much of an inconveniance for you and that you enjoy reading this issue.

We hope to have completed Issue Four by the end of March. If you would like to order copies in advance, you may do so by sending either :-
a) 70p + A4 SAE (with a 28p stamp)
b) £1.10 (which includes postage and packing)

We are very pleased to welcome Mr Bob Taylor as a contributor to the magazine and he will be helping with the programming and technical side of things.

If anybody else would like to contribute or help out with the magazine, would they please write to us at the normal address (shown below) and state what sort of things they would like to submit for publication.

If you have any question concerning the CPC, please feel free to drop us a line and we'll do our best to come up with a solution. We try to answer all letters that we receive, but please don't expect an immediate answer.

The address to write to is :-
PRINT-OUT,
8 Maze Green Road,
Bishop's Stortford,
Hertfordshire
CM23 2PJ.

MAXAM RESULT-P.17

TECHNICAL TIPS

We have received several letters asking for help concerning various types of tape/disc files. As we feel that these queries needed a fuller answer than could be given in the limited space of the letters page we have decided that a complete explanation to these, and any other large technical problems, will be answered in 'Technical Tips'. The first problem concerns MERGE and comes from MRS JO WOOD, who owns a CPC 464, 64K memory expansion and DD-1 disc drive.

DISC MERGING ON THE 464

I can't get programs to MERGE into memory. I've made sure there's no problem over line numbers etc. All that happens is every time I try to merge two programs it comes up with EOF met and when I ask the computer to print the value of EOF it comes up as -1. I don't really understand the explanation in the computer manual to be honest - "attempt has been made to read past end of file on cassette input stream" - I'm using the disc drive all this time. How can you stop it reading past the end of the file?

I suspect that you have discovered a bug in the Operating System software which resides in the 464's ROM, but there are a couple of solutions :-

1. Any program you wish to MERGE into another in memory should be saved beforehand as an ASCII file by using :- SAVE "<filename>",A
Saving in this way takes quite a bit longer but MERGE handles it OK.
2. A more expensive solution is to change your 464 into a 6128 by having a 6128 ROM fitted permanently into your computer, or by building a Romchanger similar to the project in the March 1989 issue of 'Amstrad Computer User' or you could buy a ROM board which allows the insertion of a 6128 ROM. The improvement is not astronomical but the 6128 is a lot more 'User-friendly' than the 464.

What is going wrong is as follows :-
Each location (or address) of memory can hold a value between 0 and 255. This value is called a byte. When a program is saved, the bytes which make up the program are taken in sequence from the memory and sent to the tape recorder or disc drive to be stored on the magnetic medium there. When loading, these same bytes are taken from the magnetic medium and re-stored in memory (at the same locations if it's a BASIC program being loaded). The loading process is not concerned with the values of the bytes.

When MERGEing, these same bytes are not put immediately into their final destinations. As the computer has to decide where to place each line of the program (among other things) it has to know where each line begins and ends and so looks at every byte to gather this information. In doing so, it detects any byte with the value of 26 (&1A) and because a byte of this value indicates the end of an ASCII file, it assumes it has reached the end of the program. However it knows from other information that it is not really the end and so prints the error message.

In the 6128 the fault does not occur and so I asssume that someone must have encountered the problem in the 464 ROM and written an extra check for the 6128. A BASIC program which is saved as an ASCII file is just a sequence of numbers and letters, none of which has a value of &1A. So it will have only one byte of &1A and that will be where it is supposed to be - at the end. On the other hand a BASIC program when saved normally, has almost everything converted into byte values (called TOKENS) to save room. Although only one of these tokens has the value of &1A, numbers are also converted (it is not just 26, but 282, 538, etc. or any other number made up of 26 and a multiple of 256) which are common and will cause the error. Sometimes with a short BASIC program there may be no bytes of &1A and in such cases, the MERGE will be error free. However, longer programs are almost certain to have some bytes with a value of &1A.

FILE TYPES

The next problem comes from Mr J. Hazeldine who asks :-

How does one run or list a program that you can hear loading yet will not RUN. Something must have gone into the memory because you can hear it loading, surely there must be some way of LISTing it so that it can be examined and perhaps rectified.

There is no definite answer to this as there are many reasons why a program may not allow you to list it. Before a possible solution can be suggested, you need to know what type of file you are trying to load. The simplest way of doing this is to CAT the tape. Following the filename will be a block number & then a symbol which signifies what sort of file it is. The common file types are shown below :-

$ - normal BASIC file
% - protected BASIC file
* - ASCII text file
& - binary file

If it is a normal BASIC file you should be able to load and list it quite normally unless it has some special form of protection on it. However, in order to load and list a protected BASIC file you need to use a deprotector such as that given in this issue in Deprotection . An ASCII file is usually used by a word-processor and needs to be loaded into one to view easily (or the file might be used for the MERGEing process). If the binary file is 8 blocks long it is probably a screen file and can be seen by using the command :- LOAD "<file>",&C000 If it is not this long it is probably a machine code program and, once loaded, you will need a monitor/dissassembler to view and change it. There are other files which have special symbols but the above are the most usual.

Loops :- GOTO & FOR.....NEXT

BEGINNER'S BASIC

In Issue Two we developed a program that would print an introduction message and get two numbers whose average the program would calculate and print. Below, is that program as a reminder.

```
10 CLS
20 INPUT "What is your name";name$
30 PRINT "Hello ";name$;
40 PRINT " I am your Amstrad and"
50 PRINT "I am going to calculate averages"
60 INPUT "Enter number 1 ",number1
70 INPUT "Enter number 2 ",number2
80 answer=(number1+number2)/2
90 PRINT "The average of";number1;
100 PRINT "and";number2;"is";answer
```

As it stands, this program is fine for printing out averages but there are times when you will want to calculate more than one pair of averages. Using the program above, the only way to do this is to constantly re-RUN it. However, if we could make the program run over and over again this would allow us to calculate any number of averages quickly. To do this add the following line :-

```
110 GOTO 60
```

This line incorporates a new command, GOTO. The number following GOTO tells the computer what line it should go to next. Thus the order of execution of the line numbers in the above program is as follows :-

10,20,30,40,50,60,70,80,90,100,110,60,70,80,90,100,110,60,70,etc.

Therefore, the command GOTO creates an infinite loop. The only way to break out of the above program is by pressing the [ESC] key twice - this ESCapes from the program.

GOTO is probably the most common looping instruction and also the most misused. As GOTO forms an infinite loop it is rarely used in proper programs for the purpose of looping - an important rule is that if you find yourself having to break out of a GOTO loop you should probably have used another type of loop.

The above works perfectly, but if you wanted the program to calculate ten averages only and then stop, you should not use a GOTO loop but a FOR.....NEXT loop. This command is slightly harder to understand than GOTO but is a lot less limited in its uses and abilities. On the next page are the lines that need to be modified for the program to calculate ten averages and then stop.

```
 55 FOR i=1 TO 10
110 NEXT i
```

Line 55 requires a bit of explanation - a FOR.....NEXT loop always takes the form of, FOR variable=number1 TO number2 ..... NEXT variable. The variable can be anything that you want and is used to identify which loop the NEXT command is referring to (it may help to think of it as the 'name' of the loop). The two numbers following the FOR instruction specify what value is to be given to the variable at the beginning and also what the last value is meant to be. To help explain this, type NEW and then enter the short program below :-

```
10 FOR i=10 to 20
20 PRINT i
30 NEXT i
```

What this program will do is, print all the numbers from 10 to 20. In line 10 the program is told that i must be between 10 and 20 and that it will start with a value of 10 and finish with a value of 20. Line 20 then prints the value of 'i'. Line 30 checks to see whether 'i' has reached 20 if so it ends the program but if not it adds 1 to the value of 'i' and sends it back to line 10. No doubt some people will be wondering if it is possible to make numbers increase in jumps larger than 1 - the answer is yes. Change line 10 to read :-

```
10 FOR i=10 TO 20 STEP 2
```

Now, every time line 30 is executed, it will check to see if 'i' equals 20 and if not it will increase 'i' by 2 and then go back to line 10. The STEP part of the command is optional but if it is omitted it is assumed that the variable will be INCREASED by 1 each time a NEXT command is encountered. If we want the numbers to be printed in descending order that is easy. Change line 10 to :-

```
10 FOR i=20 TO 10 STEP -2
```

The step command now tells the computer to DECREASE 'i' by 2 every time that a NEXT command is executed. If you have two loops set up at the same time, the computer will be able to distinguish between them as long as they do not have the same variable name. Tye NEW, and enter the following program :-

```
10 FOR i=1 to 10
20 FOR j=1 to 10
30 PRINT j;
40 NEXT j
50 PRINT
60 NEXT i
```

This program will print 10 rows of the numbers from one to ten with a clear line between each of them. The order of execution is quite complicated and due to lack of space cannot be printed here in full. However, if you wish to see the order of execution, type TRON (this stands for TRace ON). Now every time a line is executed, the line number is printed up on the screen. Some of these numbers may appear very quickly and you might like to pause the program by pressing [ESC] once - when you wish the program to resume, press any key. When you want to switch TRON off simply type TROFF (TRace OFF) and everything will be back to normal. TRON is very useful for finding out where mistakes or bugs are occuring in the program.

FOR ..... NEXT and GOTO are very powerful commands on their own (as are the many other types of loops) but so far we have not seen the computer make any decisions on its own. For instance, a programmer might require the computer to ask for a code number and if it is correct, to allow the program to continue, but if it is wrong to halt the program. There is one main command on the CPC, in BASIC, that is used for decision making and that is the IF....THEN....ELSE command. This is a very complicated structure (it needs to be !!!) and we will be looking at it in great detail in next issue's BEGINNER'S BASIC.

IMPROVEMENTS

Frank Ellis of Botley has sent in the following suggestions for the improvement of 'Name and Address Storer' that appeared in Issue Two of Print-Out. The alterations allow you to do two new things, and they are :-

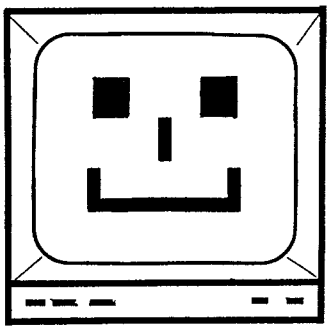
1. To enter telephone numbers in the form of 01 234 5678
2. To enter names in the form of BLOGGS, J.

To make these changes, simply alter the following lines :-

(a) Replace all occurences of phone(i) by phone$(i) in lines 340,440,590,790, 920,1040,1170,1410,1570,1740 and 2110
(b) Replace all occurences of find by find$ in lines 1550 and 1570
(c) Replace all occurences of pold by pold$ in line 2110
(d) Replace all occurences of INPUT by LINE INPUT in lines 430,460,690,990, 1490,1600,1700 and 1780
(e) Change line 870 to read :-

```
870 LOCATE 20,14:LINE INPUT "Enter correct name :- ",name$(i)
```

Linechecker



A PROGRAM TYPING AID

for use with PRINT-OUT

by BOB TAYLOR

Ever typed in a program and found that it stopped with a new error each time you tried to RUN it. We all know the feeling I'm sure. Well we at PRINT-OUT think we've got the answer for you. It is an RSX called LINECHEK which you can load in at any time - even after you've started typing, so you don't have to SAVE what you've already done while you install it.

There have been several Line Checkers in the national magazines over the years. Each gives a checksum accumulated from the characters present in a line, but the method of calculating this sum differs from magazine to magazine. Evidently it's no good just adding up the value of each character as this would give the same result even if the letters were in the wrong order - something more sophisticated is needed.

The first checker I came across was 'Get it Right' in the June 1987 issue of 'Computing with the Amstrad', as it was known then. Together with a revised version in 'CPC Computing' (as CwtA became) in September 1988, this produced a 5 digit number checksum using a fairly complex XORing routine but the big bug-bear was that you had to list the program to get the checksums - there was no way to get a result as you went along.

The most recent appearance was 'Type-writer' in 'Amstrad Action', June 1989. The checksum with this checker appears when you type in the ENTER at the end of each line and consists of a four letter code resulting from multiplying the checksum so far by the value of the current character. However the format used to present the result uses unusual letter combinations and these appear almost foreign compared to decimal or hexadecimal, as well as being difficult to read on my usual MODE 2 screen. Between these two came 'Proofreader' in 'Amstrad Computer User' also in Sept 1988 (with it's improved version appearing in Jan 1989.) As with the 'Amstrad Action' one, the checksum appears with the ENTER but it's a 2 digit hexadecimal result derived from adding in the value of each character multiplied by its position in the line, any overflow being discarded.

The important factors which make for better checking are the algorithm of the summing routine and the active size of the checksum. For these reasons, of the three listed above, the CwtA one is probably the best for eliminating coincidences (i.e. where the checksum gives the result which matches that

published by the magazine without there being many chances of an error still being present.) However I always found its method of use a big drawback and 5 digits were a bit of a mouthful too. With the demise of CwtA, this checker is now no longer used with any programs.

The next best is the 'AA' method but it falls down by making the case of letters significant so that capitals and lower case must be typed in exactly as printed. Although 'Proofreader' checksums are only 2 bytes long and its method is slightly inferior to the others, it is a lot more 'user-friendly' in that you can type in upper or lower case & it gives the same result; and it ignores accidental double spaces (which wouldn't affect the syntax of the line).

Now I could of course have written one completely from scratch which would have none of the problems present in those already produced: however I'm a great believer in standardisation and would prefer to see every magazine using the same one (hopefully mine) so I don't want to introduce more proliferation. To this end I decided to utilise 'Proofreader' for PRINT-OUT's use, with some improvements. The improvements take two forms: those which increase its user-friendliness and those which increase its discrimination. Because the latter result in a changed checksum which by definition means less standardisation, I have arranged the RSX to have three versions in one: a first which is almost identical in operation to the current 'Proofreader'; a second which mimics 'Proofreader' for lazy typists like myself; and the full blown version to be used with all future PRINT-OUT programs to help you 'get it right'. In this way it can still be used with previous ACU programs and hopefully ACU might adopt it for all their printed programs.

During the development of the routine, a major difference was discovered between the way the 464 operates the AUTO mode of program line entry and the way the 6128 does it. It had been hoped to only print checksums for lines being entered into a program whether with or without AUTO, and not with direct commands. The method employed to acheive this was to look for the line number in the BASIC Input Area (at &ACA4 in the 464 and at &AC8A in the 6128) and only to print values if it was present. Unfortunately the 464 does not have the line number copied there during AUTO (6128 owners will probably not be aware that their 464 counterparts are unable to alter the line number provided by AUTO). This gives rise to two problems. Firstly there is no number present to initiate the production of the checksum. Secondly, with no number, there are characters (digits) missing from the line, so any value produced from the rest of the line would be incorrect. In order to accommodate these differences, I had to write extra code for the 464 to enable AUTO to be used; it reconstitutes the appropriate line number from AUTO's Line Number system variable at &AC1D/E and adds in these digits to produce an accurate checksum.

I had intended to use the AUTO On flag at &AC1C to produce a checksum during use of AUTO, and not afterwards when issuing direct commands. It seems that

while in AUTO this flag is turned off temporarily during line entry and so can't be used. Instead, I decided to make the routine check the Line Number system variable: if anything other than 0 is present, then AUTO mode is assumed and a line number added into the checksum. In order to guarantee a line number of 0, any use of the RSX will reset that system variable. It will then stay as 0 until AUTO is invoked. After using AUTO, the last line number reached will still be present there, so checksums will then continue to be produced for any input unless the RSX (with or without parameters) is again used. The further problem with the 464's AUTO which only displays an asterisk when a previous line is present, cannot be accomodated and such lines will give wrong checksums while in this form; LISTing such lines produces the right values however.

All this unfortunate bodgery applies only to 464s; 6128 owners will only get checksums with program line entries - normal or AUTO.

The loader for LINECHEK, as we've called our checker, is given in the program listing. If you are typing it in then ignore the digits enclosed by squared brackets at the beginning of each line and remember to SAVE it before running it in case you have made any typing errors (the last ones to get through we hope). When RUN, it will prompt you to press 'S' in order to SAVE the Machine Code program - make sure you have a cassette or disc installed before doing so. You can then carry out your first check with LINECHEK.

You will have noticed that with the BASIC listing of LINECHEK the checksum is printed in squared brackets at the start of each program line. As mentioned earlier, this should not be typed in. The checksum will be given at the start of the line so that it is easier to check against and also to remind you not to type it in. However when you press ENTER at the end of a line, the checksum will appear after the line you have just typed in. All you have to do is check that this result is the same as PRINT-OUT's checksum printed at the beginning of that line; if not then you can EDIT the faulty line and correct the error. Of course, until you have LINECHEK installed you can't use it, so the checksums given won't be usable while you are typing it in.

To use LINECHEK now or in the future, just load the cassette or disc with the correct M/C program on and enter the following line in Direct Command Mode (i.e. without a line number):

```
MEMORY HIMEM-&238:a=HIMEM+1:LOAD"linchk.bin",a:CALL a
```

To switch on the RSX just enter:
|LINECHEK,2 (or 1 or 0 depending on which version you want to use)
To switch it off:
|LINECHEK (without any comma or number)

The three versions are:

|LINECHEK,0 This is the plain version for use with ACU programs. It does every thing that 'Proofreader' does with the added advantage that you don't have to type in a space after the line number.

|LINECHEK,1 This is the lazy man's ACU version; it performs the same as |LINECHEK,0 but with the following improvements:

a) It allows you to use ? instead of PRINT and inserts a following space if one is needed
b) It inserts a space between a decimal number and a Command if one is omitted - e.g IF a=10GOTO 100 becomes IF a=10 GOTO 100
c) It adds in (but doesn't insert) a space before & (used for Hex numbers) and before hash (used with streams) if either follows a Command without one - e.g CALL&A000 is counted as CALL &A000 and PRINT#1 as PRINT #1
d) It adds in (but again doesn't insert) a " if you've left one off the end of a line

|LINECHEK,2 This is the version used for checking all future programs printed in PRINT-OUT. It behaves like |LINECHEK,1 but discriminates better. The other versions convert all letters to lower case no matter where they are, and also ignore multiple spaces counting them as one. This version preserves the case of letters within strings enclosed by quotes as well as the number and position of spaces therein. There is a weakness in 'Proofreader' which means that it could not distinguish between upper & lower case letters in some positions in a line even if it was altered so as not to convert everything to lower case; this problem has been eliminated in this version. A further flaw in 'Proofreader' is that it mistakes a single inverted comma inside a string as a short REM. This too has been corrected for the PRINT-OUT version. As a result of these alterations, the checksums produced will differ from the other versions whenever strings are present in a line.

Note that all versions ignore everything that follows a ' (short REM,) but that everything after a REM is counted.

With the 6128 only, one way to use LINECHEK is to enter:

AUTO <first line>,5 (i.e. halve the required spacing between lines)

Type in the line intended after the line number ending in 0 and if the checksum is correct when you press ENTER then just use ENTER again when the line number ending in 5 appears - this line won't be included in the program (nor will any checksums for any lines). However if the result is wrong then change the 5 of the line number which has just appeared, to a 0 and copy from the line above correcting as you go.

Because line numbers can't be altered on the 464 during AUTO, you won't be able to use this method. If you find any incorrect results, you will either have to break out of AUTO straight away to correct a line with an incorrect

checksum, or wait until you've entered a chunk of program before EDITing the offending lines. When you LIST a program the checksum will also appear as each line is printed and this also applies to a listing printed on a printer. You don't have to POKE or alter anything to obtain printed checksums - it is done automatically within LINECHEK.

You could RUN any program with LINECHEK turned on, but if that program requires you to INPUT any numbers, (or INPUT anything on the 464) then each such entry will generate a checksum. To remove this possibility just turn it off with: |LINECHEK without any parameters - it doesn't matter which version you have been using.

|LINECHEK,1 is intended to be used with 'Amstrad Computer User' programs although unless there is a change of policy, only '10 Liners' are at present printed with checksums.

There might be a few times when ACU program writers don't leave a space between commands and & or hash. At these times |LINECHEK,0 should be used to obtain matching results, but in general |LINECHEK,1 should suffice for most ACU 'Proofreader' checked programs.

The RSX is quite flexible; if you forget which version is in use it is OK to issue a |LINECHEK,<version> again, and you can change from one version to another at any time without turning it off in between.

The final proof of the pudding is in the eating, so check |LINECHEK,2 by LISTing the BASIC program and confirming that you obtain the same checksums as are printed in the listing with this article.

```
[F1] 10 'LINECHEK-LOADER by R Taylor for PRINT-OUT (Public Domain 1989)
[1E] 20 MEMORY HIMEM-&238:RESTORE 110:PRINT:PRINT"Please wait a few seconds"
[49] 30 FOR lin=0 TO &238/8-1:total=0:FOR n=1 TO 8:READ a$
[54] 40 byte=VAL("&"+a$):POKE HIMEM+lin*8+n,byte
[4B] 50 total=total+byte:NEXT n
[21] 60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT"Error in line"lin*10+110
        :END
[14] 70 NEXT lin:IF PEEK(6)=&80 THEN POKE HIMEM+&CB,&A4:POKE HIMEM+&D4,&99:POKE
        HIMEM+&D8,&95:POKE HIMEM+&67,&21
[39] 80 PRINT:PRINT"All M/C loaded":PRINT:PRINT"Press 'S' to save M/C as
        LINCHK.BIN":WHILE INKEY$="":WEND:IF INKEY(60)<>-1 THEN a=HIMEM+1:SAVE
        "LINCHK.BIN",B,a,&238
[BA] 90 PRINT:PRINT"To Load and Initialise |LINECHEK RSX with a program present
        just Enter:":PRINT"MEMORY HIMEM-&238:a=HIMEM+1:LOAD"CHR$(34)
        "LINCHK.BIN"CHR$(34)",a:CALL a":PRINT"in Direct Command Mode with the
        Disc or Tape inserted at the correct place"
[EA] 100 END
```

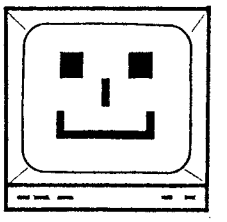
```
54] 110 DATA D5,62,6B,01,2F,02,09,EB, 2C8
F4] 120 DATA 01,19,00,09,72,2B,73,44, 177
00] 130 DATA 4D,E1,36,C9,23,C3,D1,BC, 4A0
F6] 140 DATA 2F,92,B7,20,12,3A,30,00, 214
17] 150 DATA FE,CF,20,42,2A,35,00,22, 2B0
6C] 160 DATA 2C,BD,2A,33,00,18,2E,7B, 207
1A] 170 DATA 32,37,00,3A,5A,BB,FE,CF, 385
36] 180 DATA 20,2C,2A,2C,BD,22,35,00, 1B6
8D] 190 DATA 2A,5B,BB,22,33,00,32,30, 1F7
9B] 200 DATA 00,F3,CD,0F,00,3B,3B,E1, 326
16] 210 DATA FB,01,60,00,09,22,2C,BD, 270
59] 220 DATA 0E,06,09,3E,C3,32,5A,BB, 265
94] 230 DATA 22,5B,BB,32,2B,BD,C9,00, 31B
9A] 240 DATA 00,22,1D,AC,C9,2A,1D,AC, 2A7
62] 250 DATA 7C,B5,28,67,11,F6,FF,D5, 49B
3C] 260 DATA 1E,9C,D5,11,18,FC,D5,11, 39A
C8] 270 DATA F0,D8,D5,AF,D1,3D,3C,19, 4AF
76] 280 DATA 38,FC,ED,52,B7,28,10,F6, 458
C8] 290 DATA 30,D5,0C,51,5F,78,83,15, 2D1
8E] 300 DATA 20,FC,47,7B,E6,30,D1,CB, 490
9C] 310 DATA 4B,28,E1,7D,F6,30,2E,37, 35C
1F] 320 DATA 11,A3,AC,18,2B,E5,2A,35, 2E7
68] 330 DATA 00,18,04,E5,2A,33,00,22, 180
F3] 340 DATA 31,00,F5,FE,0D,20,41,21, 2B3
F5] 350 DATA 37,00,7E,E6,03,F6,10,77, 31B
01] 360 DATA D5,11,8A,AC,C5,44,4C,1A, 38B
DF] 370 DATA FE,30,38,2A,FE,3A,30,26, 31E
16] 380 DATA C5,18,63,18,21,3E,20,F7, 2CE
7B] 390 DATA F7,3E,5B,F7,06,02,F1,4F, 3CF
CC] 400 DATA 0F,0F,0F,0F,E6,0F,C6,90, 287
F1] 410 DATA 27,CE,40,27,F7,79,10,F4, 3D0
13] 420 DATA 3E,5D,F7,3E,12,F7,C1,D1, 46B
CF] 430 DATA F1,E1,C3,30,00,18,D6,CB, 47E
37] 440 DATA 5E,CB,DE,20,04,CB,71,20, 387
43] 450 DATA 2A,CB,71,3E,70,28,26,B9, 31B
6A] 460 DATA 3E,72,28,21,B9,3E,69,28, 281
7C] 470 DATA 1C,B9,3E,6E,28,17,B9,3E, 2B7
57] 480 DATA 74,28,12,CB,9E,13,1A,FE, 342
9A] 490 DATA 2E,38,19,FE,3A,38,04,FE, 2F1
13] 500 DATA 41,38,19,3E,20,1B,F6,20, 221
9D] 510 DATA E3,2C,45,4F,7C,81,10,FD, 3AD
28] 520 DATA 67,E3,13,1A,FE,3A,30,06, 2E5
A7] 530 DATA FE,30,30,EC,CB,B6,CB,66, 4FC
BE] 540 DATA 28,06,CB,A6,FE,20,20,DB, 3B8
4A] 550 DATA B7,20,0A,CB,7E,28,0E,3E, 29E
7E] 560 DATA 22,CB,BE,18,D0,34,35,20, 31C
79] 570 DATA 26,FE,27,20,17,79,FE,20, 319
F0] 580 DATA 28,8B,18,BF,18,89,CB,4E, 344
A6] 590 DATA 28,EF,E3,2C,7D,E6,07,1A, 3AA
56] 600 DATA 20,B8,18,B5,FE,20,20,AE, 391
54] 610 DATA B9,20,AB,18,B5,18,A4,FE, 40B
DC] 620 DATA 22,20,0C,CB,7E,CB,BE,20, 340
6C] 630 DATA 9F,CB,FE,CB,B6,18,99,CB, 565
DD] 640 DATA 7E,20,D3,FE,27,28,C6,FE, 482
03] 650 DATA 3A,20,06,CB,AE,18,EC,18, 2F5
71] 660 DATA 85,06,61,FE,26,20,15,CB, 310
77] 670 DATA F6,79,F6,20,B8,38,0A,FE, 47D
D2] 680 DATA 7B,30,06,CB,6E,28,C6,CB, 3A3
F3] 690 DATA B6,1A,18,B8,CB,6E,20,08, 301
62] 700 DATA FE,3F,28,A0,FE,23,28,E1, 42F
CF] 710 DATA 1A,F6,20,B8,38,04,FE,67, 389
A3] 720 DATA 38,04,CB,B6,38,E3,FE,7B, 451
E1] 730 DATA 30,C5,7E,E6,60,20,DA,79, 42C
AA] 740 DATA FE,30,38,D5,FE,3A,38,95, 440
22] 750 DATA FE,74,20,CD,1A,F6,20,B8, 447
7B] 760 DATA 20,C7,D5,13,1A,F6,20,B8, 3B7
AD] 770 DATA 30,1A,1B,1B,1B,1A,F6,20, 1CB
D7] 780 DATA B8,20,11,1B,1A,F6,20,FE, 332
BC] 790 DATA 64,20,09,1B,1A,F6,20,B8, 290
FE] 800 DATA 30,02,CB,EE,D1,18,A2,4C, 3C2
A3] 810 DATA 49,4E,45,43,48,45,CB,00, 277
```

Linechecker

A PROGRAM TYPING AID

for use with PRINT-OUT

All future programs in Print-Out will have Linechecker codes except those that are included as part of tutorials.
With major programs, the 'Linechecker' symbol will be printed together with a paragraph which will be a reminder as to how to use it best.



HOMEBREW SOFTWARE

SHAREWATCHER II - BY MATTHEW PINDER, MIP SOFTWARE.

In our last issue another title from MiP software, Maths Master Plus, was reviewed and was highly recommended. At the end of the article it was mentioned that Sharewatcher, a stockmarket simulation, was on sale and that an upgraded program would be available soon.

Earlier this month, Sharewatcher II was released and it costs £4.50 (tape) or £7.50 (disc). Sharewatcher II is the updated version of Sharewatcher and it also features an additional printer routine.

The aim of the game is to make as much money as possible by purchasing shares of companies, preferably at a low price, and then selling them at a much higher price. However, the game doesn't just stop there. The game also gives the player updates on the shares of all 18 companies that can be traded. There is also an accounts screen which contains information about the time left, capital in shares, ready capital, total wealth and the FT-100 share index. You can also inspect your prospective investments and get an 'investment rating' which is meant to recommend which companies are worth buying. Included in the program is a printer dump which prints your accounts and shares for a permanent record.

Every so often important news appears at the bottom of the screen which will in some way affect your capital or share prices, either in general or of a particular company.

The various features of the game are well implemented and all of the information is set out in a clear manner. Unfortunately, because of this, the share prices had to be split over three screens and this was very annoying when shares were owned on several screens. The only way to buy or sell was by entering the number of the company and on these screens there was no reminder of the companies' numbers. The share price update was done only when 'U' was pressed & this further slowed the game down.

Overall, it was well thought out and the various parts of the game fitted together neatly. The first few games were quite interesting and enjoyable but it soon became tedious as one game was very similar to another.

SHAREWATCHER II costs £4.50 on tape or £7.50 on disc
Please make all cheques payable to Matthew Pinder and send to :
MiP SOFTWARE, 4 WHAM HEY, NEW LONGTON, PRESTON, LANCS. PR4 4XU.

Adventuring Games — Homebrew

from JOHN PACKHAM

PANIC BENEATH THE SEA (Price £1.99 on tape or £4.50 on disc)

This is a two part adventure written using GAC (Graphics Adventure Creator), access to the second part is via a password which is given at the end of the first part. The plot is typical. A ship, that was carrying vital defence equipment, has been sunk in the middle of the sea. You have been asked by the Government to retrieve the cargo and return it safely.

It is rare to find Homebrew games with a good loading screen but the one for Panic Beneath the Sea gives the impression of quality. Upon loading, you are given a nice, short introduction which gives you the general idea of your mission without giving too much away. The first part of the game has some locations with graphics whereas Part Two is text-only. The graphics where relevant, are clear and unfussy and, although they take a little while to appear, add greatly to the atmosphere. The text descriptions are quite detailed and there are many clues given in them. The screen layout is straightforward with no attempt to make it look particularly pleasing to the eye but as long as there are plenty of puzzles and action this is a minor concern. The game is made more lively by John Packham's own brand of jokes and the puzzles range from obvious to very tricky and the game has plenty of character interaction. Unfortunately, the parser, which is rather slow, contains some faults and accepts and understands only a small number of words (often not the most common ones).

The game contains a number of good puzzles (some easy others not) and it creates a well built-up atmosphere but the limited parser kills exploration. Not the best GAC game ever, but at the price it's not a bad buy.

CITY FOR RANSOM (Price £1.99 on tape or £4.50 on disc)

This is another game written using GAC and is slightly easier than 'Panic Beneath the Sea' and is also about half the size. City for Ransom also has a pretty loading screen and certain locations in the game have well drawn graphics which add to the atmosphere. Many of the criticisms of 'Panic Beneath the Sea' also apply to this gane, in particular the difficult parser. The object is to find a bomb, which has been hidden in the city by a terrorist who is demanding a rather large ransom, within twelve hours. This game features many good puzzles but again the parser destroys one's interst in the game after a while. However, despite this, if you are just starting to play adventures it is well worth having a look at.

PROJECT ANNIHILATION (Price £2.50 on tape or £4.50 on disc)

This adventure game was written using ADLAN (instead of GAC) and the difference shows !!! The only disadvantage to this is that there are no graphics on the tape version but there are on the disc program. The plot for this is that, a lunatic has broken into a 'top secret establishment that manufactures biological warfare weapons' and has detonated a bomb. Everyone has escaped but they have just made a new weapon which might be a bit unstable! So of you go again to save the world from disaster.

The descriptions in this game are short but informative and everything is laid out in a neat, organised and helpful manner. The parser in this game is excellent. It has a wide vocabulary of useful words, is very friendly and gives you pointers as to how you should have phrased things. By the use of the command 'WORDS' you get a short list of helpful words which can be used. The parser reacts quickly and this enables you to progress with the game easily. During the game you will come across a large number of puzzles, some of which are very devious, but all have a logical solution. It has plenty of locations to visit and things to do and when this is combined with a decent plot the game becomes rather compelling and addictive. Again the game has a beautifully drawn loading screen and even without graphics (on tape) the game seems to be much better than your average homebrew adventure. If you have the money spare, buy it. You won't be disappointed.

The games are all produced by John Packham and cost :-
PANIC BENEATH THE SEA .... £1.99 (tape) or £4.50 (disc)
CITY FOR RANSOM ..... £1.99 (tape) or £4.50 (disc)
PROJECT ANNIHILATION .... £2.50 (tape) or £4.50 (disc)
or order 2 for £3.00 (tape) or £6.00 (disc)
or order 3 for £4.50 (tape) or £9.00 (disc)
Please send all cheques/postal orders to :-
John Packham, 60 Hightown Towers, Warburton Road,
Southampton, Hants. SO2 6HH.

The WINNER of our competition from Issue Two was S. MESSINA of Heywood who will soon be receiving a copy of MAXAM for his program called 'SEABATTLE'.
The RUNNER-UP was ANTHONY MILBOURNE from Thane who will receive something for his program 'LOCO'. Both programs will appear in a future issue of Print-Out.

HOW TO PATCH THE FIRMWARE JUMPBLOCK.

BITS AND PIECES

It is often desirable to be able to change some part of the computer's firmware so that a certain function is carried out in a more efficient or different way. Fortunately, the CPC provides the user with a very easy way of incorporating new routines with the minimum of difficulty. One such method is known as 'patching the jumpblock'. Using this method, the jumpblock is altered so that when a CALL is made it is re-routed away from the routines stored in the LOWER ROM to the user's own routines which have already been assembled into the computer's memory. To fully understand how this works it is probably simplest to look at a BASIC example of the technique that we will be using.

An important part of structured BASIC programming is the subroutine. This enables a program to be constructed in a logical fashion and prevents a program for having large chunks of similar instructions. In BASIC a subroutine is called by means of the command GOSUB and the computer is then told to RETURN to the main program at the end of the subroutine. However it is sometimes necessary to have multiple subroutines (ie. subroutines which call subroutines) in order to make the program simpler and the routines more versatile. Of course the obvious and most organised way of doing this, is to use more GOSUBs to call the further subroutines.

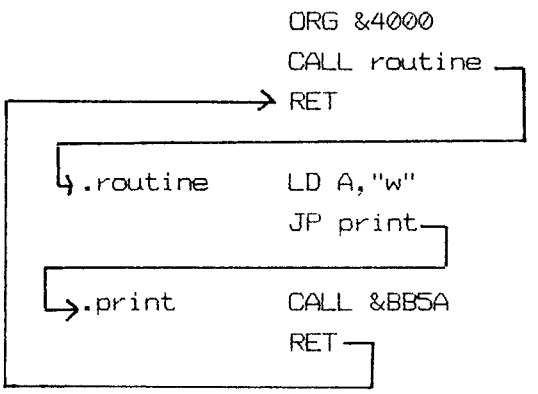
However it is sometimes impossible to do this for one reason or another and then a different method of calling multiple subroutines is required; this is shown in the example program below. The arrows illustrate the path of execution of the program.

```
10 GOSUB 1000 : REM Select name
20 ' This would really be the
30 ' rest of the program
40 END

1000 REM Select name routine
1010 a$="ARNOLD"
1020 GOTO 1500 : REM Print routine

1500 REM Print Routine
1510 PRINT a$
1520 RETURN
```

A similar program can be written in Machine Code using CALL, RETurn and JumP commands and a short example of this is shown on the next page. With a little adaption this program can be used to patch the jumpblock.



```
        ORG &4000          ; start adddress of M/C program
        CALL routine       ; calls routine
        RET                ; returns to BASIC

.routine LD A,"w"          ; A = ASCII code of 'w'
        JP print           ; jumps to second routine

.print  CALL &BB5A         ; print letter
        RET                ; return to just after the last
                           ; executed CALL command
```

The arrows indicate the path of operation through the program above.

The Jumpblock

Below is an example of how this routine can be used to change the jumpblock. This program changes any calls to SCR SET MODE (&BC0E) so that instead of changing mode this call prints the letter 'm'. The first thing to do is to assemble the piece of alternative code that will actually do the printing. The necessary code is shown below and starts at &4000.

```
ORG &4000          ; start address of program
LD A,"m"           ; A = ASCII code of 'm'
CALL &BB5A         ; print the letter
RET                ; return
```

We now need to alter the jumpblock entry so that it re-routes any calls to &BC0E away from the firmware routine in the lower ROM to the pre-written code at &4000. The way to do this is to change the bytes at &BC0E/&BC0F/&BC10 to :- &C3,&00,&40 respectively (JP &4000). Now any calls to &BC0E or MODE commands will be sent to &4000 and the letter 'm' will be printed.

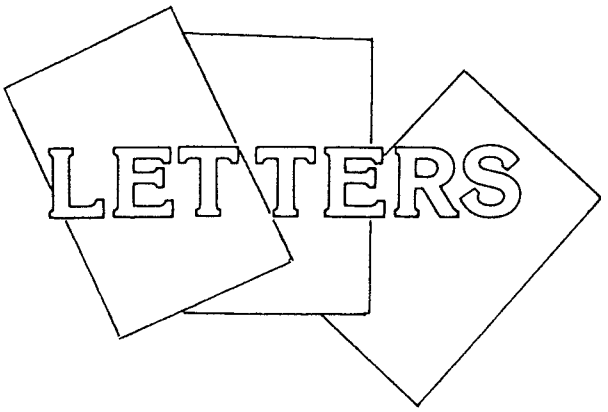
The way in which this relates to the above section on multiple subroutines is not entirely obvious. In your program you have either CALL &BC0E or a MODE (which is the same thing). This sends the program to the jumpblock (the first subroutine) which JumPs (the same as GOTO) to the second subroutine which is our code. At the end of our code is a RETurn which sends the program back to the command which immediately follows the CALL or MODE command.

To restore everything to normal you only need to enter the following line :-

```
CALL &BD37
```

This resets the entire jumpblock to its original values.

In this column we give your thoughts an airing, try to solve your problems and give you the chance to speak to other CPC owners. If you have got something to say about the CPC. write to the usual address.



Printer Problems - DMP 2000

I am hoping that you or one of your readers may be able to help me over a problem with my DMP 2000 printer in getting it to print graphics. (I have a CPC 464 with expanded memory and disc drive.) In the printer manual itself there is a graphics screen dump routine with an example program and I have tried this but all I get is one line of smudge and nothing else.

Yet, my printer DOES print graphics as I have 'Stop Press' and have no trouble with printing from that. I am wondering if it's something to do with the DIP switch settings but I tried altering them recently but only ended up stopping the thing printing text as well!! It is working OK now except for this graphics problem. I have just got Dk'tronics graphics lightpen system and the printer dump for Amstrad printers with that also doesn't work with my DMP 2000!

Mrs Jo Wood
ROCHDALE

Unfortunately I do not own a DMP 2000 and cannot offer a solution to your graphics problem but if any of our readers do know the answer we would be very grateful if they could get in touch with us so that we can pass it on to Mrs Wood. However, I have heard of the problem with the Dk'tronics lightpen and the solution is to use the screen dump for EPSON printers as opposed to the one for AMSTRAD printers. This is because the Amstrad printer referred to is the old DMP 1 which had a set of non-standard control codes. All other Amstrad printers (including the DMP 2000) use standard Epson codes and so this dump should be used.

Machine Code Queries

I have tried to learn Machine Code many times, but I find the authors of some seldom cater for the raw beginner. Often simple explanations are never made and I would like to name one or two.

1) Machine Code programs don't seem to have line numbers, yet it is never stated thus.
2) I suppose the (second) right hand column ; are REM statements or do they have to be there for the program to run.
3) Some statements have an underline in them (eg. test_key) - does this need to be entered as well.
4) Does upper case or mixed case make any difference or should it be strictly adhered to.
5) How to use an assembler would be much appreciated.

I suppose I have entered into computing too late in life, but I find it very exciting and rewarding and also a very good pastime in retirement.

J. Hazeldine
LONGBRIDGE

In answer to your questions about Machine Code programming :-

1. When a M/C program is entered into an assembler it is typed in without line numbers in the form in which it appears in Print-Out. What the assembler then does is to translate your program into pure numbers which it then pokes into the correct space in memory. As the numbers are poked directly into memory and any jumps are made to an address (in memory) no line numbers are needed.
2. The columns following the semi-colon (;) are indeed comment columns and anything following the semi-colon can be ignored.
3. Most assemblers allow you to enter names or LABELS which can be used in a similar way to variables in BASIC. However, in M/C you cannot have spaces in the labels and so the underline (_) character is a standard way of replacing the space that cannot be included.
4. The use of upper and lower case depends on your assembler. Some are more tolerant than others but generally it does not matter what case is used.
5. Again, because there are so many assemblers on the market it would be impossible to explain how to work each one in this magazine but if you have a specific problem with any assembler we will do our best to solve it.

Cassettes versus Discs

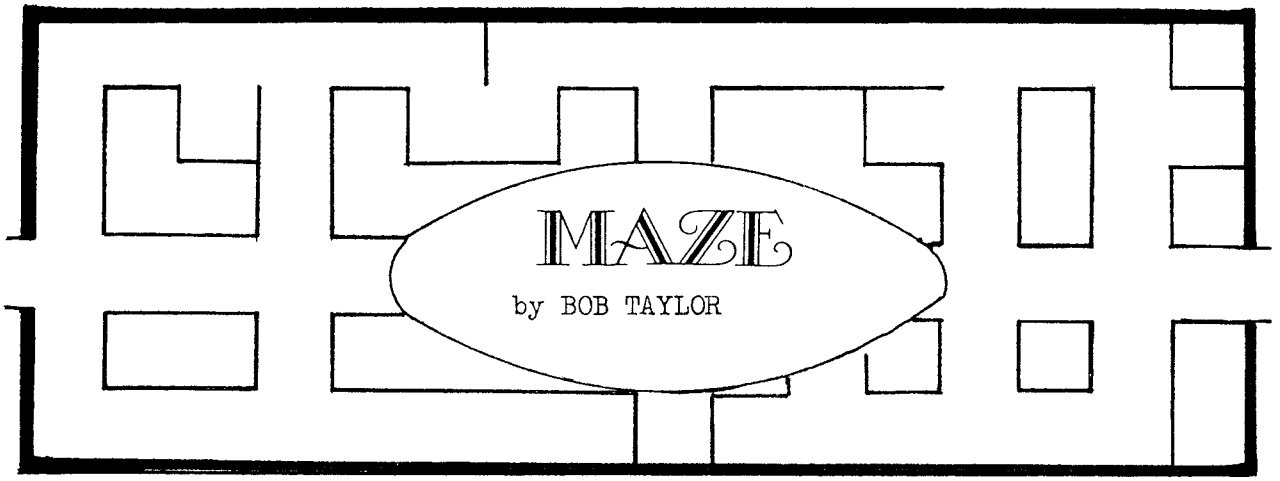
I think that Print-Out is one of the best magazines around, mainly because it contains so many useful articles on programming. I saw an advert for it in 'Amstrad Action' and decided to give it a try. Because of the very reasonable price I decided I wouldn't be loosing much if it turned out to be rubbish. I think the idea of selling cassettes with each issue's programs on, is great & I would certainly be prepared to pay £1.00 for a few programs I wanted. If you recorded how many copies of each program you sold, you could do a bumper cassette every six months or so, containing all the most popular ones. You could do another competition with the prize being a cassette containing as many programs (from Print-Out) as the winner can fit on it. I think that lots of competitions with small prizes are better than a couple with huge prizes because more people can win. If you do find a shop with firmware manuals still in stock, you should buy a few and sell the spare ones through the magazine, I bet you'd get loads of orders (mine for a start). I am luck enough to own a 6128 and Cassette Unit but lots of people just have 6128s and by making everything in the magazine cassette based you are losing business. I would like to see articles on :- How to use the expansion port on a 6128 to control robots, Modems - the pros and cons, Hacking and Artificial intelligence.

Anthony Milbourne
THAME

I'm glad you like the idea of selling Program Cassettes because we are now offering a special bargain to those of you who have missed earlier issues of Print-Out. You can now obtain the Program Cassette and copy of the magazine for each of the back issues for just £1.75 including postage and packing. You can also obtain the programs for each issue on disc for 50p providing you include a standard Amsoft 3 inch disc with your order. If any of our readers does know of a shop which still has Firmware Manuals in stock please get in touch with us as we know of many people who would be interested in buying one.

All of your suggestions for future articles have been taken into account and if we receive enough requests for them we may include them in future issues.

If you have any comments on anything to do with the Amstrad CPC, then please write to us at :- PRINT-OUT, 8 Maze Green Road, Bishop's Stortford, Hertfordshire CM23 2PJ.



The object is to start at one side of a maze and find your way to the exit at the other side. You decide how large you want the maze to be and then a clever routine makes a random path through it. Side chambers are created from the unused parts and finally a few extra walls are removed to make the route even more complicated. In addition a diamond is placed in any corner of the maze which is not an entrance or an exit. Each diamond carries a bonus which reduces your total number of moves. However you don't have to collect bonuses and could decide to avoid them if at all possible.

If you think the maze drawn is too difficult, you can press the 'A' key and another of the same size will be drawn. Otherwise press SPACE or any other key when you are ready and you will enter the maze and be presented with a perspective view from where you stand. Movement is by means of the UP cursor key and each move is counted, even when you walk into the walls! The other cursor keys can be used to turn LEFT, RIGHT or AROUND but don't count as moves. The exit & entrance are shown as large crosses, but once inside the maze you cannot leave again via the entrance.

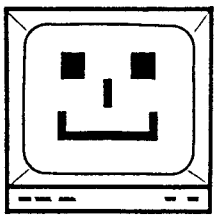
Each diamond present appears on the far wall of its room and can be collected by just walking into the room. This earns you the bonus and a free look at the plan of the maze showing your position and the direction you are facing. Once collected, the diamond disappears. To return to the maze, just press any key. The bonus cannot take your move score to less than zero, so if your score when you get there would be reduced to below zero, it might be worth planning your route so as to get the full bonus by going there later. The bonus is the square root of the total size of the maze.

If you get lost you can see the plan by pressing 'M', although this will result in a penalty of extra moves (as large as the bonus) being added to the score. Once out of the maze, the plan is shown together with the paths you took (all those false moves are shown too), but with no penalty of course.

You can then choose to go through the same maze again from the entrance by pressing 'R', either to use the program as a multi-player game, or to try to lower your own score, or you can start another maze of either the same size by pressing 'A' or one of a completely new size by pressing any other key.

Following the program listing are comments on its workings and the various techniques that the program employs.

PROGRAM:



```
[F1] 10 '  MAZE  by Bob Taylor for PRINT-OUT (copyright 1989)
[F2] 20 '*** Set up ***
[FF] 30 MEMORY &3FFF:RANDOMIZE TIME:DEFINT a-z:m(0)=0
[0C] 40 SYMBOL 248,0,0,32,124,32:SYMBOL 249,0,16,56,16,16,16:SYMBOL 250,0,0,8,
        124,8:SYMBOL 251,0,16,16,16,56,16
[44] 50 SYMBOL 252,0,0,0,0,0,0,0,1:SYMBOL 253,1,1,1,1,1,1,1,1:SYMBOL 254,0,0,0,
        0,0,0,0,255:SYMBOL 255,1,1,1,1,1,1,1,255
[FD] 60 FOR n=&8000 TO &801E:READ i$:POKE n,VAL("&"+i$):NEXT
[36] 70 DATA 3e,40,cd,08,bc,21,00,40,11,00,c0,44,4d,ed,b0,3e,c0,c3,08,bc,21,00,
        c0,11,00,40,42,4b,ed,b0,c9
[6F] 80 MODE 1:INK 0,13:INK 1,0:WINDOW #1,8,33,1,25
[F9] 90 '*** Make Maze ***
[B1] 100 LOCATE 1,24:INPUT"Maze size, width (5 to 36) ";w:IF w<5 OR w>36 GOTO
         100
[C5] 110 LOCATE 11,25:INPUT"height (5 to 20) ";h:IF h<5 OR h>20 GOTO 110
[46] 120 CLS:INK 2,10:INK 3,12:ERASE m:DIM m(2,w+1,h+1):c=20-INT((w+1)/2):r=11-
         INT((h+1)/2)
[B1] 130 ORIGIN 319-16*INT((w+1)/2),224+16*INT((h+1)/2)
[12] 140 FOR a=0 TO h:MOVE 0,-16*a:DRAW 16*w,-16*a:NEXT
[D9] 150 FOR n=0 TO w:MOVE 16*n,0:DRAW 16*n,-16*h:NEXT:ORIGIN 320,200
[1C] 160 FOR y=0 TO h:m(1,0,y)=9:FOR x=0 TO w:m(0,x,y)=255:NEXT x:m(1,x,y)=9:
         NEXT y
[F3] 170 FOR y=0 TO h+1 STEP h+1:FOR x=0 TO w+1:m(1,x,y)=9:NEXT x,y
[08] 180 y=1+INT(RND*h):LOCATE c,y+r:PRINT">";:x=1:sy=y:m(0,x-1,y)=252:m(1,1,y)
         =4:n=0
[B3] 190 '*** Make path through Maze ***
[C4] 200 n=n+1:d=2+((d=1 AND RND<0.8 OR d=2 AND RND(0)<0.45)AND y<>1)-((d=3 AND
         RND(0)>0.2 OR d=2 AND RND(0)>0.55)AND y<>h)
[00] 210 y=y+(d=1):m(0,x,y)=m(0,x,y)AND (&FE-(d AND 1))
[48] 220 IF x<>w AND n=6 THEN n=0:m(0,x,y)=m(0,x,y)AND &FE
[4C] 230 LOCATE x+c,y+r:PRINT CHR$(m(0,x,y));
[7E] 240 y=y-(d=3):x=x-(d=2):m(1,x,y)=d+2:IF x<>w+1 GOTO 200
[84] 250 LOCATE x+c,y+r:PRINT">";:ey=y:d=0:GOTO 280
[AE] 260 '*** Make side chambers ***
[59] 270 PAPER 3:m(0,x+(d=0),y+(d=1))=m(0,x+(d=0),y+(d=1))AND(&FE -(d AND 1))
[65] 280 LOCATE x+c,y+r:PRINT CHR$(m(0,x,y));:IF x=1 AND y=sy GOTO 380
[D0] 290 dx=(d=0)-(d=2):dy=(d=1)-(d=3):x=x+dx:y=y+dy:IF m(1,x,y)=0 THEN
         m(1,x,y)=d+2
[F5] 300 fw=m(1,x+dx,y+dy)=0:IF fw AND RND<0.5 GOTO 270
[D6] 310 sw=m(1,x+dy,y-dx)=0:IF sw AND RND<0.6 THEN d=(d+3)MOD 4:GOTO 270
[80] 320 IF m(1,x-dy,y+dx)=0 THEN d=(d+1)MOD 4:GOTO 270
[C9] 330 IF sw THEN d=(d+3)MOD 4:GOTO 270
```

(continued on the next page...)

(continued from previous page)

```
[99] 340 IF fw GOTO 270
[D2] 350 IF m(1,x-dx,y-dy)=0 THEN d=(d+2)MOD 4:GOTO 270
[55] 360 PAPER 2:IF m(1,x,y)=0 THEN m(1,x,y)=d:d=m(1,x+dx,y+dy)MOD 4:GOTO 280
         ELSE d=m(1,x,y)MOD 4:GOTO 280
[B1] 370 '*** Make extra openings ***
[B1] 380 PAPER 0:FOR n=1 TO w*h/20:x=INT(1+RND*(w)):y=INT(1+RND*(h)):a=INT
         (253.5+RND*1.5)
[24] 390 IF (a<>254 OR x<>w)AND(a<>253 OR y<>h)AND m(0,x,y)<>(m(0,x,y)AND a)
         THEN m(0,x,y)=m(0,x,y)AND a:LOCATE x+c,y+r:PRINT CHR$(m(0,x,y))ELSE
         n=n-1
[C6] 400 NEXT:FOR n=1 TO 3000:NEXT:CALL &8014:INK 2,13:INK 3,3
[5A] 410 FOR a=1 TO h:FOR n=1 TO w:m(1,n,a)=&80:NEXT n,a:x=1:y=sy:d=2:mov=0
[56] 420 n=INT(SQR(w*h)):m(2,1,1)=-n*(sy<>1):m(2,1,h)=-n*(sy<>h):m(2,w,1)=-n*
         (ey<>1):m(2,w,h)=-n*(ey<>h)
[FA] 430 LOCATE 11,24:PRINT"Press SPACE to start":LOCATE 5,25:PRINT"(or A to
         create a different maze)"
[E1] 440 m(1,x,y)=m(1,x,y)OR(&10-(d=3)-2*(d=0)-4*(d=1)-8*(d=2))
[26] 450 LOCATE x+c,y+r:PRINT CHR$(22)CHR$(1)CHR$(d+248)CHR$(22)CHR$(0);
[CC] 460 i$=UPPER$(INKEY$):IF i$=""GOTO 460 ELSE IF i$="A"GOTO 120:ELSE a=-3*
         (i$=CHR$(242))-(i$=CHR$(243))-2*(i$=CHR$(241)):IF a THEN d=(d+a)MOD 4
[B2] 470 '*** Print Compass and Menu ***
[D5] 480 CLS:PEN 3:LOCATE 1,6:PRINT"F'wrd "CHR$(240)TAB(37)"N"TAB(1)"Left  "
         CHR$(242)
[5B] 490 PRINT"Right "CHR$(243)TAB(35)"W   E"TAB(1)"About "CHR$(241)
[5F] 500 PRINT"Map   M"TAB(37)"S":LOCATE 35,15:PRINT"MOVE":PEN 1
[A7] 510 '*** Draw view inside Maze ***
[69] 520 CLS#1:tx=x:ty=y:s=200:da=d AND 1
[8C] 530 FOR n=1 TO MIN(14,-((w+1-x)*(d=2)+x*(d=0)+(h+1-y)*(d=3)+y*(d=1))):
         ps=s:s=0.8*s:psw=-ps:prw=ps
[E9] 540 IF m(2,x,y)THEN IF n=1 THEN CALL &8000:PEN 3:LOCATE 14,24:PRINT
         MIN(m(2,x,y),mov)"MOVE BONUS":mov=MAX(0,mov-m(2,x,y)):LOCATE 10,25:
         PRINT"press any key for maze";:PEN 1:m(2,x,y)=0:GOTO 440
[C5] 550 IF m(2,x,y)THEN MOVE 0,s/2:DRAW -s/2,0:DRAW 0,-s/2:DRAW s/2,0:DRAW 0,
         s/2
[EE] 560 ex=n:fw=m(0,x+(d=0),y+(d=1))AND(1+da):IF fw THEN n=14 ELSE IF d=2 AND
         x=w OR d=0 AND x=1 THEN ex=0
[49] 570 IF m(0,x+(d=1),y+(d=2))AND(2-da)THEN sw=ps ELSE sw=s:IF d=1 AND x=1 OR
         d=3 AND x=w THEN psw=-s ELSE IF NOT m(0,x+(d<2)-(d=3),y+(d=1 OR d=2)
         -(d=0))AND(1+da)THEN IF fw THEN psw=-s-1 ELSE psw=-INT(0.98*ps)
[1A] 580 IF m(0,x+(d=3),y+(d=0))AND(2-da)THEN rw=ps ELSE rw=s:IF d=3 AND x=1 OR
         d=1 AND x=w THEN prw=s ELSE IF NOT m(0,x+(d=0 OR d=3)-(d=1),y+(d<2)
         -(d=2))AND(1+da)THEN IF fw THEN prw=s+1 ELSE prw=INT(0.98*ps)
[F2] 590 MOVE psw,sw:DRAW -s,s:DRAW -s,-s:DRAW psw,-sw
```

(continued on the next page...)

(continued from previous page)

```
[A3] 600 IF psw=-s THEN MOVE -ps,ps:DRAW -s,-s:MOVE -ps,-ps:DRAW -s,s ELSE IF
     psw=-INT(0.98*ps) THEN DRAWR 0,2*s ELSE IF psw=-s-1 THEN IF d=2 AND
     x=w OR d=0 AND x=1 THEN MOVE -1.1*s,0.9*s:DRAW -ps,0.75*s:MOVE -1.1*s,
     -0.9*s:DRAW -ps,-0.75*s
[D2] 610 MOVE prw,rw:DRAW s,s:DRAW s,-s:DRAW prw,-rw
[AB] 620 IF prw=s THEN MOVE ps,ps:DRAW s,-s:MOVE ps,-ps:DRAW s,s ELSE IF prw=
     INT(0.98*ps) THEN DRAWR 0,2*s ELSE IF prw=s+1 THEN IF d=2 AND x=w OR
     d=0 AND x=1 THEN MOVE 1.1*s,0.9*s:DRAW ps,0.75*s:MOVE 1.1*s,-0.9*s:
     DRAW ps,-0.75*s
[F4] 630 x=x+(d=0)-(d=2):y=y+(d=1)-(d=3):NEXT:x=tx:y=ty
[E7] 640 IF ex THEN MOVE -s,s:DRAW s,s:MOVE -s,-s:DRAW s,-s ELSE s=0.9*s:MOVE
     -s,s:DRAW s,-s:MOVE -s,-s:DRAW s,s:IF x<>1 AND d=0 OR x<>w AND d=2
     THEN ex=n-1
[B4] 650 LOCATE 37,8:PRINT CHR$(248+d):LOCATE 35,17:PEN 3:PRINT mov:PEN 1
[B2] 660 '*** Get move ***
[5C] 670 i$=UPPER$(INKEY$):IF i$=""GOTO 670
[ED] 680 a=-3*(i$=CHR$(242))-(i$=CHR$(243))-2*(i$=CHR$(241)):IF a THEN d=(d+a)
     MOD 4:GOTO 520
[71] 690 IF i$="M"GOTO 810
[F2] 700 IF i$<>CHR$(240)GOTO 670:ELSE m(1,x,y)=m(1,x,y)OR(&10+d-(d=3)-8*(d=0))
[7D] 710 IF ex>1 THEN x=x+(d=0)-(d=2):y=y+(d=1)-(d=3):m(1,x,y)=m(1,x,y)OR(&10-
     (d=3)-2*(d=0)-4*(d=1)-8*(d=2)):mov=mov+1:GOTO 520
[35] 720 IF ex OR x=1 THEN LOCATE 13,25:PRINT LEFT$("ENTRANCE CLOSED",-15*
     (ex=0));:SOUND 1,480,50:FOR a=1 TO 1000:NEXT:mov=mov+1:PRINT CHR$(17):
     PEN 3:GOTO 650
[AD] 730 '*** Escape ***
[13] 740 mov=mov+1:PEN 3:LOCATE 35,17:PRINT mov:LOCATE 14,22:PRINT"OUT IN "mov
     "MOVES":FOR n=1 TO 1500:NEXT
[24] 750 CALL &8000:LOCATE 14,22:PRINT"OUT IN "mov"MOVES":PEN 1:LOCATE 6,23:
     PRINT"press R for repeat of this maze,":LOCATE 6,24:PRINT"A for
     another of the same size":LOCATE 7,25:PRINT"or any other key for new
     maze";
[6F] 760 PEN 3:PRINT CHR$(22)CHR$(1);:FOR a=1 TO h:FOR n=1 TO w:IF m(1,n,a)<>
     &80 THEN LOCATE n+c,a+r:PRINT CHR$(m(1,n,a));
[24] 770 IF INKEY$<>"" THEN n=w:a=h
[8C] 780 NEXT n,a:PEN 1:PRINT CHR$(22)CHR$(0);
[34] 790 i$=UPPER$(INKEY$):IF i$=""GOTO 790 ELSE CLS:IF i$="R"THEN CALL &8000:
     GOTO 410 ELSE IF i$="A"GOTO 120 ELSE 80
[A8] 800 '*** Show plan of Maze ***
[D0] 810 CALL &8000:IF mov>0 THEN a=INT(SQR(w*h)):mov=mov+a:LOCATE 12,24:PRINT
     a"MOVE PENALTY"
[E8] 820 LOCATE 9,25:PRINT"press any key for maze":GOTO 450
```

MAZE - NOTES

The program is typical of Spectrum BASIC programs with much manipluation of seemingly unrelated numeric variables to achieve results. An array with three dimensions is used (the three dimensions are not the three parts about to be described. For that, one of the diemensions is divided into three). One part of the array holds the ASCII codes of User Defiened Characters which represent the walls present for each square of the maze (the initial drawing of the maze grid is done using graphics and not printing so as to speed up its presentation). A second part is used firstly during the path routine to hold a value which represents the direction from which that square was entered. This is then used to retrace the paths back to the entrance during side-chamber creation. During movement through the maze, this part is used to hold the player's path which will be displayed when the maze is exited. The last part of the array holds the bonuses you could receive from collecting diamonds.

A typical technique is to operate on numbers using the fact that a False condition evaluates to a 0, while a True means -1. On the Spectrum, True would have been 1 and this is very convenient for use in this way. On the CPC, the negative value, whilst being essential for use with the logical operators AND, OR and XOR (the Spectrum only had AND and OR and they were not used for logic) does mean that all signs refering to conditions have to be inverted, making for further obscurity in an already complex looking program. If you can work through to discover what the computer is doing, you could be rewarded with ideas to use in your own programs.

M/C ROUTINE

A short Machine Code routine is poked into memory. The CPCs are quite slow at printing characters on the screen, so to avoid a slow build-up of the plan of the maze, a copy is kept in the lower screen area (at &4000 to &7FFF) from where the routine re-copies it to the normal (upper) screen at &C000 to &FFFF. To see the upper screen while this is happening would show a sort of 'Venetian Blind' effect due to the way that screen RAM is laid out. To eliminate this, the screen is switched to lower before the copy and then back to upper afterwards. The visual effect is an instantaneous showing of the plan. Compare this to the time taken to PRINT the route taken, especially on larger mazes.

BASIC DEPROTECTION

PROGRAMMING UTILITY —

We have received several requests for us to print a program that will remove the protection on a BASIC program that has been protected using the command :-

SAVE "<filename>",P

Files that have been saved by this command can then only be loaded and run by using the command :- RUN "<filename>"

This means that the program cannot be loaded, listed, altered or saved without first removing this protection. The program printed below does remove this protection and allow you to study or alter the program. However as most commercial software is not written in BASIC and is also protected by highly complex loader programs, this program will not allow you to break into and hack proper games.

Following the program are notes on what it does and how, a brief explanation of what the CPC does when it protects a BASIC (or a Machine Code) program and instructions on how to use the program.

Program

```
[F1] 10 'DEPRO-LOADER copyright R Taylor 1989
[39] 20 RESTORE 110:PRINT:PRINT"Please wait a few seconds"
[7E] 30 FOR lin=0 TO &40/8-1:total=0:FOR n=0 TO 7:READ a$
[C4] 40 byte=VAL("&"+a$):POKE &BE80+lin*8+n,byte
[4B] 50 total=total+byte:NEXT n
[0D] 60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT"Error in line"lin*10+110
        :PRINT:END
[7F] 70 NEXT lin:IF PEEK(6)=&80 THEN POKE &BEB3,&45
[62] 80 PRINT:PRINT"All M/C loaded":PRINT:PRINT"Press 'S' to save M/C as DEPRO.
        BIN":WHILE INKEY$="":WEND:IF INKEY(60)<>-1 THEN SAVE "DEPRO.BIN",B,
        &BE80,&40
[76] 90 PRINT:PRINT"To Load and Initialise DEPRO just Enter:":PRINT"LOAD"CHR$
        (34)"DEPRO.BIN"CHR$(34)":CALL &BE80":PRINT"in Direct Command Mode with
        the Disc or Tape inserted at the correct place":PRINT"To switch off
        just Enter CALL &BE80,0"
[EA] 100 END
```

(continued on next page)

(continued from previous page)

```
[2B] 110 DATA B7,3A,7A,BC,20,1C,FE,C3, 424
[3A] 120 DATA C8,32,BB,BE,2A,7B,BC,22, 3F6
[2E] 130 DATA BC,BE,3E,C3,21,AD,BE,F5, 4FC
[EC] 140 DATA E5,32,7A,BC,22,7B,BC,E1, 487
[D2] 150 DATA F1,C9,FE,C3,C0,3A,BB,BE, 5EE
[D3] 160 DATA 2A,BC,BE,18,EA,CD,A5,BE, 4D6
[97] 170 DATA F5,AF,32,2C,AE,F1,CD,7A, 4E8
[09] 180 DATA BC,18,D7,00,00,00,00,00, 1AB
```

To use the above program, first type it in exactly as it appears (ignore the numbers in square brackets at the beginning of the line, these are LINECHECKER codes and should not be typed in. They are there to help you spot any typing errors that may occur) then run the program. When prompted insert a tape or disc to which the Machine Code is to be saved for future use. If you wish to use the program in the future you simply have to load and initialise this code simply use :- LOAD "DEPRO.BIN":CALL &BE80
You can then load the program to be deprotected in the normal way. When it is loaded you will find that you are able to list and alter the program.

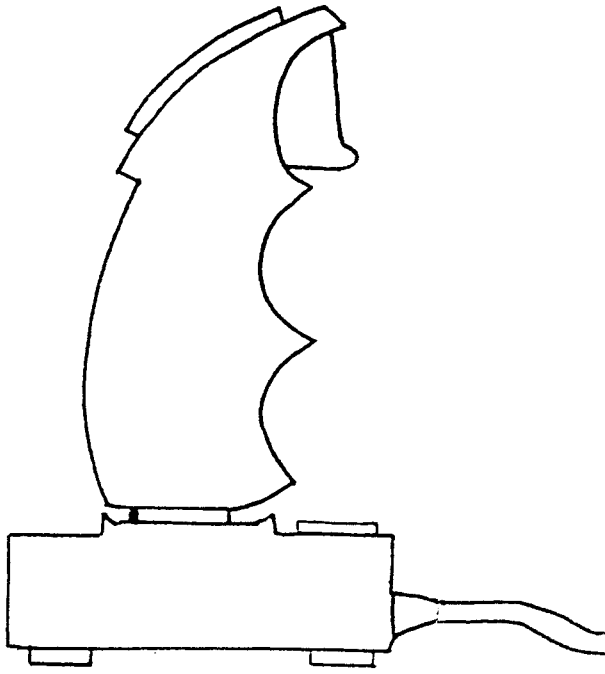
EXPLANATION

At &AE2C on the 6128 (or &AE45 on the 464) there is a system flag which is set when a protected BASIC program is detected during loading. This flag is tested after LOADing and, if it is set, the whole of the Free Space area of memory is wiped clean including the BASIC program area, thus preventing LISTing of the program. However, this flag is not checked after RUN or CHAIN and this explains why the program can be loaded and run with these commands.

The 'deprotecter' works by intercepting the call to CAS/DISC IN CLOSE at &BC7A after loading, and resetting the previously set flag to &00 before continuing with the close routine.

The program is protected by XORing its bytes with a sequence of &80 random bytes before SAVEing. During the loading process, each byte is XORed again with the same byte from the sequence so restoring the original value. The sequence does not seem to be present anywhere in ROM or RAM, so must itself be derived from existing bytes somehow. The sequence must be the same for all versions of the CPC as a program SAVEd on a 464 can be LOADed and RUN on a 6128.

GAMES REVIEWS



Fantasy World Dizzy from Codemasters

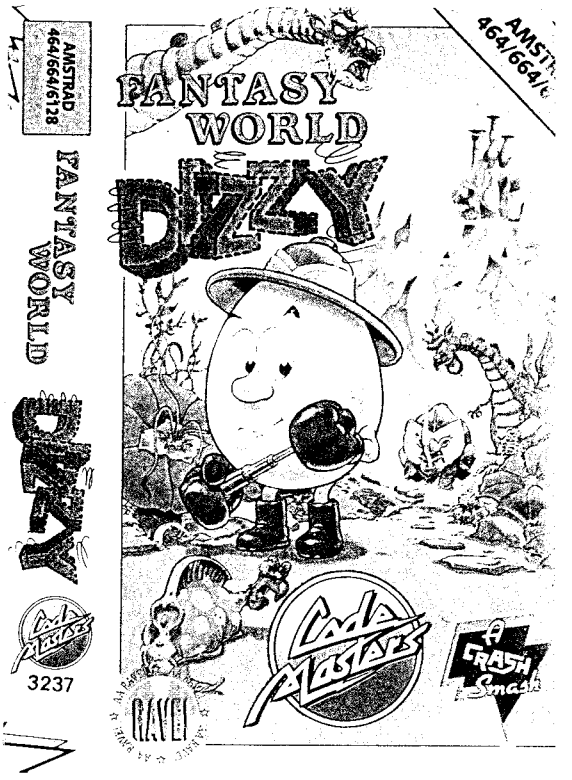
FANTASY WORLD DIZZY is the third of Dizzy's adventures from Codemasters and costs £2.99 on tape only. As with all of the games in this series, you control Dizzy, a cute little egg, who in this particular game must defeat the Evil King who is holding his 'egg-friend', Daisy. The plot is a standard one - Daisy has been imprisoned in the Wizard Weird's Tallest Tower in his Cloud Castle. Dizzy was thrown in the King's Dungeons and must rescue Daisy by using the many items that he finds throughout 'Fantasy World'. However, even when he has managed to free Daisy there is a further mission - Daisy orders him to find 30 Gold Coins so that they can buy their own house and live happily ever after. The game is not nearly as bad as the storyline and contains many unusual and funny features which add to the atmosphere.

The game takes place in 'Fantasy World' and boasts over 50 locations including volcanoes, dragons' lairs, mysterious new worlds, palaces and finally the Cloud Castle itself. Each of these locations is named and contains detailed and colourful graphics which draw you into the fantasy world of Dizzy and the other Yolkfolk that you meet on your travels. The only real weak point of the game was the sound - the music lacks any tune whatsoever and slows down to a drone in some rooms - and there are no sound effects throughout the game at all, but if you turn the volume off you should be able to forget all about the sonix and enjoy the game.

Dizzy is incredibly well animated and rolls, leaps, tumbles and falls in a most convincing and amusing manner. The various monsters which he must defeat in the game are also well thought out and designed as are the Yolkfolk from whom he gains valuable information and items of equipment. The Yolkfolk are probably the greatest addition to this game and the clever and witty conversations that the hero has with them are a pleasure to read. Each of them has an individual style of action ranging from the lazy Dozy to the historical Grand-Dizzy that helps to make the game far more interesting than other games of the same type.

The game differs from Treasure Island Dizzy by allowing you to hold upto five items (if you have the bag) and the use/drop menu enables you to make sure that the right item is used at the right time. You also have three lives which are needed to complete it safely. The game contains many nice additions to the game that increase its playability such as when Dizzy has drunk a bottle of whisky and starts making random rolls in all directions. The game is both joystick & keyboard controlled (although you cannot redefine the keys they are easy to use and well positioned).

The game is very easy to get into and highly addictive as there are many obstacles that need to be passed by utilising the numerous objects. Unlike some games it is not hard to get started but contains enough tricky sections to keep you playing for a long time. Apart from the sound, there are no real problems and it is an entertaining game to have and at £2.99 it represents excellent value for money.



MARK

I really enjoyed this game. The graphics were excellent and the playablility was tremendous but the sound was abysmal. I found the pick up and drop menu was a bit confusing at first but on the whole it was of a very high standard. It is the best Dizzy adventure yet. I'd recommend it to anyone. 83

JON

The graphics in Fantasy World Dizzy were well drawn but many of them were quite similar with little variation. The puzzles in the game were well thought out and had logical solutions but were possibly a bit too easy. The music was terrible but this did not spoil the game too much. A good buy !!! 87

TOM

The game, although in design very like previous ones, has been put together very well with clear graphics, tricky puzzles, funny characters to find and is extremely playable. However it may be a bit easy to complete and the sound can easily be improved on. Still a fantastic game to have. 80

Grid Iron II from

GRID IRON 2 is written by Alternative Software and costs £2.99 on tape only. In the game you play the part of a manager of an American Football team who must successfully guide his team through the 15 games of the season and finally win the Superbowl itself. The game allows you to trade players, select your team, play the game or even borrow money from a bank. All choices are made from a series of menus and various information such as injuries to players and finance are also displayed when relevant. Once you have selected your team and elected to play the game, you are shown a spectator's view of the stadium and can see the highlights of the game as blocky figures run about the playing area. At the end of the match you are shown the results of all the other teams and then you repeat the sequence to play another game.

Unfortunately, the game is terrible to play. It is written mainly in BASIC, with a machine code routine tacked on to allow you to see the games, and it is full of bugs. The menus take an age to be displayed and many of the options do not work as they are supposed to and this causes great confusion. On example of this is when your name changes randomly during the season. The skill and energy indicators for each player in your team have no effect and again these change for no apparent reason. At certain points in the game the computer crashes and informs you that there is a 'SYNTAX ERROR in Line ...' No matter how well designed a game is, problems like these are bound to destroy its appeal.

MARK

This game should never have been released ! The gameplay is dull and it is incompetently programmed (and in BASIC, too !!!) There is no sound except a cheer and a beep as the results are printed. The graphics are small and uninteresting. Avoid this game if you can !!! 37

JON

The bugs in this game are atrocious and the sound is non-existent. The game graphics were quite well done but soon became boring as all of the moves were identical. The game was also far too easy and became very tedious. 46

TOM

This game looks as though it is 5 years old, the sound and graphics are terrible. I lost all interest in it before I had finished a season ! 41

MACHINE CODE-

ARITHMETIC

In our last issue, we showed you how to print either single letters or whole messages and introduced the concept of subroutines. In this issue we will cover some examples of simple arithmetic in machine code and how to show the results.

I always feel that the best way of learning something is to set yourself a task involving the topic you are studying and try to arrive at a solution for that problem. In this case, we shall try to write a program that will accept 2 numbers from the user, check to see whether they are less than 5 (but more than zero), and then calculate their sum and print it on the screen.

The best way of writing any program (but especially in Machine Code) is to write small modules that carry out certain functions and then fit them together to form a complete program.

The first module involves getting a number from the operator and printing it on the screen. We shall use several subroutines so that they can be re-used as often as possible. To input a number (or letter for that matter) we need to use a new firmware call. There are about six CALLs in the Firmware Manual that look promising but the one that is best suited to our needs is KM_WAIT_CHAR which is at address &BB06. This CALL waits until a key is pressed and when one is, the A register holds the ASCII code of the character entered. This CALL however, does not then print the character to the screen. If we were using BASIC the computer would automatically accept a character and then print it on the screen, but in M/C, we must do even the simplest task ourselves. So, we then need to call TXT_OUTPUT (&BB5A) to do this. The small routine below which we have called .input. will do this for you.

```
                ORG &4000
                .input
CD 06 BB        CALL &BB06      : input character (A contains ASCII code)
CD 5A BB        CALL &BB5A      : print character whose ASCII code is in A
C9              RET             : return to BASIC
```

We now need to print a '+' sign, input another number and print an '=' sign. To do this we will use the same routine as we developed above but it will be turned into a subroutine which can be used to input both numbers without having to repeat the same chunk of code. The routine below will do this but it will need a considerable amount of alteration before it can be put to the final use that we want it for.

```
                ORG &4000
CD 11 40        CALL input      ; goto 'input' subroutine
3E 2B           LD A,43         ; A = ASCII code of a '+' sign
CD 5A BB        CALL &BB5A      ; print '+" sign
CD 11 40        CALL input      ; goto 'input' subroutine
3E 3D           LD A,61         ; A = ASCII code of an '=' sign
CD 5A BB        CALL &BB5A      : print '=' sign
C9              RET
                .input
CD 06 BB        CALL &BB06      : input character
CD 5A BB        CALL &BB5A      : print character
C9              RET             ; return from subroutine
```

That is all the work on the presentation that we need to do but now we have to tackle the more complicated part - doing the calculations. Unlike BASIC, we cannot just use the inputted numbers to do our sums but first need to convert them into numbers that the computer will accept. There are two problems - the first is easy to explain and quite simple to solve whereas the second is rather more tricky.

1. If you follow the path the program above takes whilst being executed, it should be fairly obvious that the A register, which is used to store the value of the inputted number, is also used for other things. For this reason the A register can only be used as a temporary store for the number's value. The easy solution is to copy the contents of the A register to another register after the routine, .input, has been executed and then to copy it back when we are ready to do the calculations. For this we will use the B and D registers.

2. We have already mentioned that after .input has been called, the A register contains the ASCII code of the number that the user entered. However, this code is not the value of the number in decimal. For example if the user pressed the '1' key then A would contain the number 49 and this could not be used for arithmetic. If you look carefully at the numbers and their ASCII codes a simple solution should appear. By subtracting 48 (&30) from the number's ASCII code, you get the real value of the number.

Once we have the actual values of the numbers in B and D, and the sum has been printed out, it is very easy to do the addition. The program below does all that is necessary providing B and D contain the correct values.

```
        ORG &4000
78      LD A,B          ; let A = the contents of B
82      ADD A,D         ; add D to A and store the result in A
C9      RET
```

Of course before the answer can be printed we have to convert the real value back into a printable ASCII code number. To do this we just add &30 to A. The answer can then be printed using a normal call to &BB5A. The program below is the completed program so far. In it there is one mnemonic which you may not recognize, that is SUB but all it does is subtract the value following it from the value in the A register (with both ADD and SUB the answer is stored in A). ADD and SUB are explained more fully after the program.

```
                ORG &4000       ; start address of program
CD 1E 40        CALL input      ;
D6 30           SUB &30         ; convert value in A to proper value
47              LD B,A          : store value in B
3E 2B           LD A,43         ;
CD 5A BB        CALL &BB5A      : print '+" sign
CD 1E 40        CALL input      ;
D6 30           SUB &30         ; convert value in A to proper value
57              LD D,A          : store value in D
3E 3D           LD A,61         ;
CD 5A BB        CALL &BB5A      ; print '=' sign
78              LD A,B          ;
82              ADD A,D         ; do the addition
C6 30           ADD A,&30       ; convert number to its ASCII code
CD 5A BB        CALL &BB5A      ; print the answer
C9              RET             ; return to BASIC
                .input
CD 06 BB        CALL &BB06      ; input character
CD 5A BB        CALL &BB5A      ; print character
C9              RET             : return from subroutine
```

Both the ADD and SUB commands work only on A out of the 8 bit registers. So it is not possible to have the command SUB B,D. This can be quite restrictive but as you become more skilled you will be able to find ways round any problems that you meet. The ADD command needs you to specify that it is the A register you wish to alter (eg. ADD A,34) but with SUB it is assummed (eg.SUB 34).

This routine doesn't work for numbers larger than nine or smaller than zero, or even for sums which have an answer that lies outside these two values. If you enter two numbers which have values that when summed come to more than nine you will get various symbols for your answer. In our instructions as to what we were to do, we were told to check whether they fell between zero and five and if not, to get another entry. In order to do this we shall need to modify our input routine as follows :-

```
                .input
CD 06 BB        CALL &BB06      ; get the number
FE 30           CP 48           : subtract 48 from it but don't change A
FA 1E 40        JP M,input      ; jump to .input if it is negative (ie. <48)
FE 39           CP 57           : subtract 57 from it but don't change A
F2 1E 40        JP P,input      ; jump to .input if it is positive (ie. >57)
CD 5A BB        CALL &BB5A      ; print the value
C9              RET             ; return from subroutine
```

This will allow you to enter only 0, 1, 2, 3 or 4 and so the answer cannot go above eight. The two instructions, JP M,input and JP P,input are known as conditional jumps and are rather complicated. They will be explained in the next issue but for now you will have to take them on trust. This prevents strange symbols being printed in place of numbers, but is rather limiting. Most calculations that are done come to more than nine and as this program cannot handle them something more complex is required.

At some time or other you will need to print a large number. The numbers you will want to print from machine code, whether they are scores for a new game or number of records entered in a Database, will be stored in a single register, a register pair or perhaps stored in the memory. Numbers up to 255 can be stored in a single byte (or register) while those up to 65535 will require two bytes with the low byte first if it is stored in memory (eg. the number &C145 would be stored in memory as 45 C1). You must remember however, that a printed number always has the digits representing higher multiples of 10 first.

In our next issue we will include a routine that will print larger numbers, a section which explains conditional jumps and the FLAG (F) register. In the meantime you could try to adapt the program above, so that it would calculate both the sum of two numbers and also the result when one number is subtracted from the other. The above routines contain all the information you need to be able write such a program.

BASIC Poker

Next to all the routines contained in this issue (in the leftmost column) there are rows of numbers. These numbers should not be typed into an ASSEMBLER but are intended for use by those who do not possess one. These numbers can be typed into the 'BASIC Poker' which was printed in both Issues One and Two. The 'BASIC Poker' is included on every issue's program cassette and also includes instructions on how to use it correctly.

Advanced Basic

-STRINGS-

In this issue we are going to look at the various text handling commands, which can perform quite varied and complicated things. The simplest of these allows you to enter a word (or letter) and check to see whether one particular action should be done next or not. Then you can write a routine which lets the user select which action, out of a number of possibilities should be done. As well as these, there are many other useful ideas which can be inserted into a larger program.

The easiest one is the simple check to see if a particular key is pressed. This involves the command INKEY$ and the routine below will check to see if any key is pressed :-

```
10 PRINT "Press any key to continue"
20 a$=INKEY$:IF a$="" THEN GOTO 20
30 END
```

If a key is pressed, then INKEY$ returns the key's value which is then stored in a$ (this is then checked against an empty string and, if it is not equal, allows the program to continue). If we wished to check to see if a particular key was pressed we would just have to modify 30 to read :-

```
30 IF a$="Y" THEN END ELSE GOTO 20
```

This now checks to see if the Y key is pressed. Another way of doing this is :-

```
20 a$=INKEY$:IF a$<>"Y" THEN GOTO 20
30 END
```

If the user entered 'y' instead of 'Y', the computer would not accept this and proceed as if another letter had been pressed. One way round this is :-

```
10 PRINT "Press 'Y' to continue"
20 a$=UPPER$(INKEY$):IF a$<>"Y" THEN GOTO 20
30 END
```

What this does is to convert whatever key is pressed into its capital value and then store this in a$ before carrying on with the program in the same way as is explained above. The opposite of UPPER$ is LOWER$ and has a similar use.

More often than not, you will want the user to make a choice from a number of different options and then execute the correct instructions. Shown below is a simple routine that will do just this.

```
 10 PRINT "1. Say Hello"
 20 PRINT "2. Open book"
 30 PRINT "3. Read book"
 40 PRINT "4. Shut book"
 50 PRINT "5. Say Goodbye"
 60 PRINT "Please enter your choice"
 70 know$="12345"
 80 where=0
 90 a$=INKEY$:IF a$="" THEN 90
100 where=INSTR(know$,a$)
110 IF where=0 then 90
120 ON where GOSUB 1000,2000,3000,4000,5000
130 END
```

Lines 10 - 60 show you the options available and prompt you to enter a number from 1 to 5. Line 70 stores the numbers/letters that the program will accept in know$. Line 80 sets a variable, where, to 0 and Line 90 inputs a character. Line 100 is slightly different - what it does it to see whether the character in a$ (the one entered) is in the string, know$. If isn't then 'where' holds 0 and line 110 checks for this and then inputs another letter. If it is, 'where' holds the position that a$ occurs in know$. This may sound a bit confusing but if you look at the example below, everything should be clearer.

If the number '2' was entered then a$="2"
Then the computer checks to see if '2' is held in know$ (know$="12345")
It is, so it counts each letter in know$ until it finds the character '2'
It finds that '2' is the second number in the string and so where=2

Therefore, in line 120 it GOSUBs to the second number in the sequence, which happens to be line 2000.

The real subroutines at lines 1000,2000,3000,4000 and 5000 have not been included in the above program as it is only an example listing. However in a real program these lines would contain the actual routines which perform the various options offered. The above program relies on the 'INSTR' command to work and is very easy to change so that it will select more or less functions - to do this, all you have to do is change line 70 to include other letters which the program is to accept and line 120 to send the program to its different subroutines.

So far we have only looked at INKEY$, UPPER$, LOWER$ and INSTR but there are several other commands which are commonly used involving strings - such as :- LEFT$, RIGHT$, MID$ and LEN. All of these are fairly straightforward and simple to use.

LEFT$ - This often takes the form b$=LEFT$(a$,1) and this command takes the first letter in the string a$ and stores it in b$. If the number in brackets is changed to 3. then the first three letters of a$ will be stored in b$.

RIGHT$ - This has the same syntax as LEFT$. eg. b$=RIGHT$(a$,2) - This takes the last two letters of the string, a$. and stores them in b$.

MID$ - This is slightly more complicated but takes a similar form to both of the commands above. eg. b$=MID$(a$,3,2) - This takes the next two letters of a$ starting from the third letter and stores this in b$. If the second number is omitted then the string extends to the end of a$.

LEN - This is different from the others in that it returns the length of the string. eg b=LEN(a$) - This will store the length of a$ in 'b'.

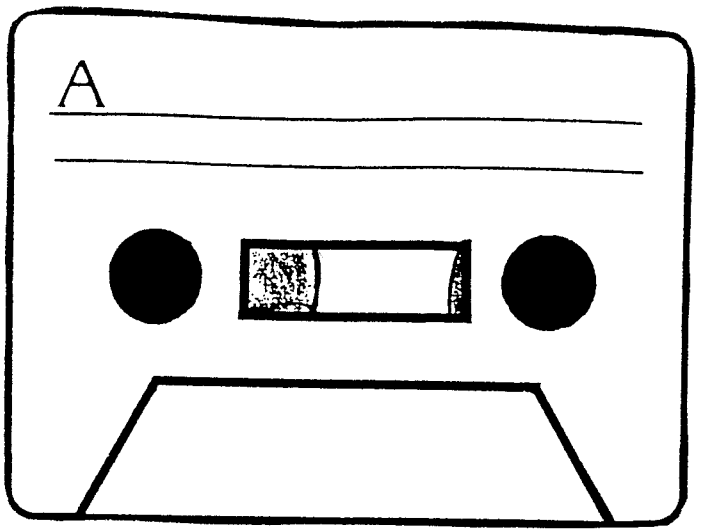
The short program below illustrates the principles of these commands :-

```
10 a$="PRINT-OUT"
20 b$=LEFT$(a$,2):PRINT b$
30 c$=RIGHT$(a$,3):PRINT c$
40 d$=MID$(a$,3,4):PRINT d$
50 e=LEN(a$):PRINT e
```

The program below does not contain any more new commands but uses the ones which we have already looked at to obtain a rather more complicated result. The program asks you for a name, such as Joe Bloggs, and then automatically abbreviates it to 'J. Bloggs' which is then printed onto the screen. It does not matter how many names are entered (it will accept Joe Harold Bloggs) and still abbreviate it correctly. Try and read through the program to discover how it actually works and you may well be rewarded with a few interesting ideas of use.

```
 10 REM Name abbreviator
 20 MODE 2
 30 PRINT "Please enter the name to be abbreviated :-"
 40 INPUT ">>> ",name$
 50 name$=UPPER$(name$)
 60 a=INSTR(name$," ")
 70 IF a=0 THEN PRINT name$:END
 80 b$=MID$(name$,1,a-1)
 90 in$=LEFT$(b$,1)
100 in$=in$+". "
110 PRINT in$;
120 name$=MID$(name$,a+1)
130 GOTO 60
```

OFFERS



Please make all cheques/postal orders payable to 'Print-Out'. Orders will be dealt with on a first-come first-served basis, although program cassettes and back issues can be guaranteed. Please do NOT send cash unless it cannot be avoided.

Please send your orders to :- PRINT-OUT, Special Offers, 8 Maze Green Road, Bishop's Stortford, Hertfordshire CM23 2PJ.

PROGRAM CASSETTES

If you wish to receive a tape containing all the programs in this issue and a booklet that explains how they and others work, please send either :-

a) A blank tape (15 minutes) + 50p (p+p)

or b) £1.00 (which includes tape and p+p)

The program cassettes for Issues One and Two are still available at the same price but also see below for a special offer on back issues and program tapes.

BACK ISSUES

We have a limited number of copies of Issues One and Two still available and further copies of Issue Three can be ordered. The price is £1.10 and this includes postage and packing (alternatively send 70p and an A4 SAE with a 28p stamp). Also see below for a special offer on back issues and program tapes.

ISSUE 4

If you wish to order Issue Four in advance please send a cheque for £1.10 (or 70p and an A4 SAE with a 28p stamp) to the usual address and it will be sent to you as soon as it published.

SPECIAL OFFER

This issue's special offer will especially appeal to those of you who have only recently started reading Print-Out. If you order a copy of any back issue of Print-Out and a program cassette the cost is only £1.75 including postage and packing and also the cost of the tape. Alternatively you can send £1.25 plus a tape of at least fifteen minutes for a back issue and program cassette.

PROGRAM DISCS

As a trial for this month, we will also be supplying the programs on disc. The cost for this is 50p and a blank disc per issue. Unfortunately we cannot offer a price inclusive of the disc at present. If enough people order using this service it may become a regular offer.

SMALLADS

FOR SALE :- LORDS OF MAGIC adventure game featuring over 70 detailed text locations. Includes 'Talk' command and 'Cast' for spells. Send cheque/postal order for £3.95 (Amstrad CPC disk) or SAE for details to :-
T. KINGSMILL SOFTWARE, 202 Park Street Lane, Park Street, St. Albans, Hertfordshire AL2 2AQ.

Remember you can place an advertisement of upto 40 words (including name and address) in this section of the magazine, free of charge. For larger advertisements please write for details and prices.

WRITING FOR PRINT-OUT

If you would like to write for Print-Out in any way, then please get in touch with us at our usual address :- PRINT-OUT, 8 Maze Green Rd, Bishop's Stortford, Hertfordshire CM23 2PJ. Any articles, programs, etc. that you would like to be considered for publication in Issue Four should reach us by no later than the 20th March 1990. Thank you.